PEC2 - Análisis de Datos Ómicos

F. JAVIER MORILLA JIMÉNEZ

2025-05-18

Table of Contents

1. Introducción y objetivos

La pandemia de COVID-19, causada por el virus SARS-CoV-2, ha generado un interés sin precedentes en la caracterización de la respuesta del huésped a la infección. En este contexto, el estudio realizado por McClain et al. utilizó secuenciación de ARN (RNA-Seq) para analizar muestras de sangre periférica de sujetos con COVID-19, comparándolas con sujetos afectados por coronavirus estacional, gripe, neumonía bacteriana y controles sanos. Los datos generados por dicho estudio están disponibles públicamente en el repositorio Gene Expression Omnibus (GEO) bajo el identificador GSE161731.

El presente trabajo tiene como objetivo principal llevar a cabo un análisis de expresión génica diferencial a partir de los datos generados por McClain et al., centrándonos en tres cohortes específicas: COVID19, Bacterial y Healthy. Para ello, se realizará un análisis exhaustivo siguiendo los pasos descritos a continuación:

- **Carga de datos**: Se descargarán los datos de expresión génica desde el repositorio GEO y se cargarán en el entorno de trabajo.
- **Preprocesamiento**: Se llevará a cabo un preprocesamiento de los datos, que incluirá la normalización y la eliminación de genes con baja expresión.

- **Análisis de expresión diferencial**: Se realizará un análisis de expresión diferencial utilizando el paquete DESeq2, que es ampliamente utilizado para este tipo de análisis en datos de RNA-Seq.
- **Visualización de resultados**: Se generarán gráficos para visualizar los resultados del análisis de expresión diferencial, incluyendo gráficos de dispersión y mapas de calor.
- **Interpretación biológica**: Se llevará a cabo una interpretación biológica de los resultados obtenidos, incluyendo la identificación de vías y procesos biológicos enriquecidos en los genes diferencialmente expresados.
- **Conclusiones**: Se presentarán las conclusiones del análisis y se discutirán las implicaciones de los resultados en el contexto de la respuesta inmune al SARS-CoV-2.
- **Referencias**: Se incluirán las referencias bibliográficas relevantes para el análisis realizado.

2. Obtención y preparación de datos

En esta sección, se procederá a la obtención de los datos de expresión génica desde el repositorio GEO y a su preparación para el análisis. Se utilizará el paquete `GEOquery` para descargar los datos y el paquete `DESeq2` para realizar el análisis de expresión diferencial.

```
library(GEOquery)
```

```
## Cargando paquete requerido: Biobase
```

```
## Cargando paquete requerido: BiocGenerics
```

```
##
## Adjuntando el paquete: 'BiocGenerics'
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, saveRDS, setdiff,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
```

```
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Setting options('download.file.method.GEOquery'='auto')

## Setting options('GEOquery.inmemory.gpl'=FALSE)

# Descargar los datos de expresión y metadatos
gse <- getGEO("GSE161731", GSEMatrix = TRUE)

## Found 1 file(s)

## GSE161731_series_matrix.txt.gz

expr_data <- exprs(gse[[1]])
metadata <- pData(gse[[1]])
expr_data

##      GSM4913486 GSM4913487 GSM4913488 GSM4913489 GSM4913490 GSM4913491
##      GSM4913492 GSM4913493 GSM4913494 GSM4913495 GSM4913496 GSM4913497
##      GSM4913498 GSM4913499 GSM4913500 GSM4913501 GSM4913502 GSM4913503
##      GSM4913504 GSM4913505 GSM4913506 GSM4913507 GSM4913508 GSM4913509
##      GSM4913510 GSM4913511 GSM4913512 GSM4913513 GSM4913514 GSM4913515
##      GSM4913516 GSM4913517 GSM4913518 GSM4913519 GSM4913520 GSM4913521
##      GSM4913522 GSM4913523 GSM4913524 GSM4913525 GSM4913526 GSM4913527
##      GSM4913528 GSM4913529 GSM4913530 GSM4913531 GSM4913532 GSM4913533
##      GSM4913534 GSM4913535 GSM4913536 GSM4913537 GSM4913538 GSM4913539
##      GSM4913540 GSM4913541 GSM4913542 GSM4913543 GSM4913544 GSM4913545
##      GSM4913546 GSM4913547 GSM4913548 GSM4913549 GSM4913550 GSM4913551
##      GSM4913552 GSM4913553 GSM4913554 GSM4913555 GSM4913556 GSM4913557
##      GSM4913558 GSM4913559 GSM4913560 GSM4913561 GSM4913562 GSM4913563
##      GSM4913564 GSM4913565 GSM4913566 GSM4913567 GSM4913568 GSM4913569
##      GSM4913570 GSM4913571 GSM4913572 GSM4913573 GSM4913574 GSM4913575
##      GSM4913576 GSM4913577 GSM4913578 GSM4913579 GSM4913580 GSM4913581
##      GSM4913582 GSM4913583 GSM4913584 GSM4913585 GSM4913586 GSM4913587
##      GSM4913588 GSM4913589 GSM4913590 GSM4913591 GSM4913592 GSM4913593
##      GSM4913594 GSM4913595 GSM4913596 GSM4913597 GSM4913598 GSM4913599
##      GSM4913600 GSM4913601 GSM4913602 GSM4913603 GSM4913604 GSM4913605
##      GSM4913606 GSM4913607 GSM4913608 GSM4913609 GSM4913610 GSM4913611
##      GSM4913612 GSM4913613 GSM4913614 GSM4913615 GSM4913616 GSM4913617
##      GSM4913618 GSM4913619 GSM4913620 GSM4913621 GSM4913622 GSM4913623
##      GSM4913624 GSM4913625 GSM4913626 GSM4913627 GSM4913628 GSM4913629
##      GSM4913630 GSM4913631 GSM4913632 GSM4913633 GSM4913634 GSM4913635
##      GSM4913636 GSM4913637 GSM4913638 GSM4913639 GSM4913640 GSM4913641
##      GSM4913642 GSM4913643 GSM4913644 GSM4913645 GSM4913646 GSM4913647
##      GSM4913648 GSM4913649 GSM4913650 GSM4913651 GSM4913652 GSM4913653
##      GSM4913654 GSM4913655 GSM4913656 GSM4913657 GSM4913658 GSM4913659
##      GSM4913660 GSM4913661 GSM4913662 GSM4913663 GSM4913664 GSM4913665
##      GSM4913666 GSM4913667 GSM4913668 GSM4913669 GSM4913670 GSM4913671
##      GSM4913672 GSM4913673 GSM4913674 GSM4913675 GSM4913676 GSM4913677
##      GSM4913678 GSM4913679 GSM4913680 GSM4913681 GSM4913682 GSM4913683
```

Se ha probado a descargar los datos directamente con el paquete GEOquery y el identificador GEO. Sin embargo, algo fallaba en el proceso de descarga, y la matriz de expresión estaba vacía. Así que hemos descargado directamente la matriz de expresión y los metadatos y los hemos cargado.

```
library(readr)

# Cargar la matriz de expresión
exprs_data <- read_csv("GSE161731_counts.csv.gz", col_names = TRUE)

## New names:
## Rows: 60675 Columns: 202
## ── Column specification
## ──────────────────────────────────────────────────────── Delimiter:
"," chr
## (1): ...1 dbl (201): 94189, DU09-03S0000604, DU09-03S0000611, 105920,
## DU09-03S0000774,...
## ℹ Use `spec()` to retrieve the full column specification for this
data. ℹ
## Specify the column types or set `show_col_types = FALSE` to quiet this
message.
## • `` -> `...1`

# Cargar los metadatos
metadata <- read_csv("GSE161731_counts_key.csv.gz", col_names = TRUE)

## Rows: 198 Columns: 9
## ── Column specification
──────────────────────────────────────────────────────────
## Delimiter: ","
## chr (8): rna_id, subject_id, age, gender, race, cohort,
time_since_onset, ho...
## dbl (1): batch
##
## ℹ Use `spec()` to retrieve the full column specification for this
data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet
this message.

# Convertir exprs_data a data.frame
exprs_data <- as.data.frame(exprs_data)
```

Ahora, verificamos la estructura de los datos de expresión y los metadatos.

```
dim(exprs_data)

## [1] 60675   202

head(exprs_data)
```

```
##              ...1 94189 DU09-03S0000604 DU09-03S0000611 105920 DU09-
03S0000774
## 1 ENSG00000223972     0               0               3     49
6
## 2 ENSG00000227232   393             142             152    246
586
## 3 ENSG00000278267     0              22              29     44
104
## 4 ENSG00000243485     0               0               0      0
1
## 5 ENSG00000274890     0               0               0      0
0
## 6 ENSG00000237613     0               0               0      0
0
##   DU09-03S0000775 DU09-03S19478 DU14-03S0000878 DU14-03S0000889 DU18-
02S0011619
## 1              46            35              26              14
0
## 2             570           213             559             369
192
## 3              69            58              73              56
0
```

```
head(metadata)
```

```
## # A tibble: 6 × 9
##   rna_id      subject_id age   gender race  cohort time_since_onset
hospitalized
##   <chr>       <chr>      <chr> <chr>  <chr> <chr>  <chr>
<chr>
## 1 94189       A1BD46     57    Female Blac… Bacte… <NA>
<NA>
## 2 DU09-03S00… 44DF6B     19    Male   Blac… Influ… <NA>
<NA>
## 3 DU09-03S00… 658A11     14    Male   White Influ… <NA>
<NA>
## 4 105920      61DE97     21    Female White Influ… <NA>
<NA>
## 5 DU09-03S00… 4D4F7C     50    Female Blac… Influ… <NA>
<NA>
## 6 DU09-03S00… C4D511     39    Female Blac… Influ… <NA>
<NA>
## # ℹ 1 more variable: batch <dbl>
```

```
dim(metadata)
```

```
## [1] 198   9
```

Vemos que la matriz de expresión tiene 60.675 genes y 202 muestras, y la tabla de metadatos tiene 198 muestras y 9 variables.

Asignamos ahora los nombres de las filas de la matriz de expresión a los nombres de los genes para que sea más fácil trabajar con ellos.

```
rownames(exprs_data) <- exprs_data[, 1]
head(rownames(exprs_data))
```

```
## [1] "ENSG00000223972" "ENSG00000227232" "ENSG00000278267"
"ENSG00000243485"
## [5] "ENSG00000274890" "ENSG00000237613"
```

Como hemos visto antes, el número de muestras en los metadatos y en la matriz de expresión no coincide. Vamos a filtrar los datos para que contengan solo las muestras que están en ambos conjuntos de datos.

```
common_samples <- intersect(metadata$rna_id, colnames(exprs_data))
exprs_data <- exprs_data[, common_samples]
metadata <- metadata[metadata$rna_id %in% common_samples, ]
head(exprs_data)
```

```
##                 94189 DU09-03S0000604 DU09-03S0000611 105920 DU09-
03S0000774
## ENSG00000223972     0               0               3     49
6
## ENSG00000227232   393             142             152    246
586
## ENSG00000278267     0              22              29     44
104
## ENSG00000243485     0               0               0      0
1
## ENSG00000274890     0               0               0      0
0
## ENSG00000237613     0               0               0      0
0
##                 DU09-03S0000775 DU09-03S19478 DU14-03S0000878 DU14-
03S0000889
## ENSG00000223972              46            35              26
14
```

Ahora vemos que el objeto `exprs_data` tiene 198 columnas que corresponden con el mismo número de muestras que los metadatos.

Teniendo los datos de expresión y los metadatos, vamos a crear un objeto `SummarizedExperiment` que es el formato adecuado para trabajar con datos de expresión en R.

```
library(SummarizedExperiment)
```

```
## Cargando paquete requerido: MatrixGenerics
```

```
## Warning: package 'MatrixGenerics' was built under R version 4.4.2
```

```
## Cargando paquete requerido: matrixStats

## Warning: package 'matrixStats' was built under R version 4.4.2

##
## Adjuntando el paquete: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians

##
## Adjuntando el paquete: 'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars

## The following object is masked from 'package:Biobase':
##
##     rowMedians

## Cargando paquete requerido: GenomicRanges

## Cargando paquete requerido: stats4

## Cargando paquete requerido: S4Vectors

##
## Adjuntando el paquete: 'S4Vectors'

## The following object is masked from 'package:utils':
##
##     findMatches

## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
```

```
## Cargando paquete requerido: IRanges

## Warning: package 'IRanges' was built under R version 4.4.2

##
## Adjuntando el paquete: 'IRanges'

## The following object is masked from 'package:grDevices':
##
##     windows

## Cargando paquete requerido: GenomeInfoDb

## Warning: package 'GenomeInfoDb' was built under R version 4.4.2
```

```
se <- SummarizedExperiment(
    assays = list(counts = as.matrix(exprs_data)),
    colData = metadata
)
assay(se, "counts")[1:10, 1:5]
```

```
##                 94189 DU09-03S0000604 DU09-03S0000611 105920 DU09-
03S0000774
## ENSG00000223972     0               0               3     49
6
## ENSG00000227232   393             142             152    246
586
## ENSG00000278267     0              22              29     44
104
## ENSG00000243485     0               0               0      0
1
## ENSG00000274890     0               0               0      0
0
## ENSG00000237613     0               0               0      0
0
## ENSG00000268020     0               0               0      0
0
## ENSG00000240361     0               0               0      7
0
## ENSG00000186092     0               0               0      0
0
## ENSG00000238009   169              72              64     31
17
```

```
head(colData(se))
```

```
## DataFrame with 6 rows and 9 columns
##                           rna_id  subject_id         age      gender
##                      <character> <character> <character> <character>
## 94189                      94189      A1BD46          57      Female
## DU09-03S0000604 DU09-03S0000604      44DF6B          19        Male
## DU09-03S0000611 DU09-03S0000611      658A11          14        Male
```

```
## 105920                   105920      61DE97          21      Female
## DU09-03S0000774 DU09-03S0000774      4D4F7C          50      Female
## DU09-03S0000775 DU09-03S0000775      C4D511          39      Female
##                                 race      cohort time_since_onset
##                           <character> <character>      <character>
## 94189           Black/African American   Bacterial               NA
## DU09-03S0000604 Black/African American   Influenza               NA
## DU09-03S0000611                 White   Influenza               NA
## 105920                          White   Influenza               NA
## DU09-03S0000774 Black/African American   Influenza               NA
## DU09-03S0000775 Black/African American   Influenza               NA
##                 hospitalized     batch
##                  <character> <numeric>
## 94189                     NA         1
## DU09-03S0000604           NA         2
## DU09-03S0000611           NA         2
## 105920                    NA         1
## DU09-03S0000774           NA         1
## DU09-03S0000775           NA         1
```

Además, hemos verificado que el objeto `SummarizedExperiment` tiene las dimensiones correctas y que los metadatos están correctamente asignados a las muestras.

Ya hemos confirmado que ambas tablas contienen los mismos genes y muestras. A continuación, vamos a usar EnsEnsDb.Hsapiens.v86 para obtener la información de los genes y añadir las coordenadas de los genes.

```
library(EnsDb.Hsapiens.v86)
```

```
## Cargando paquete requerido: ensembldb
```

```
## Cargando paquete requerido: GenomicFeatures
```

```
## Cargando paquete requerido: AnnotationDbi
```

```
## Cargando paquete requerido: AnnotationFilter
```

```
##
## Adjuntando el paquete: 'ensembldb'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
library(GenomicRanges)

gene_ranges <- genes(EnsDb.Hsapiens.v86)
head(gene_ranges)
```

```
## GRanges object with 6 ranges and 6 metadata columns:
##                    seqnames      ranges strand |         gene_id
```

```
gene_name
##                      <Rle>   <IRanges>  <Rle> |     <character>
<character>
##   ENSG00000223972        1 11869-14409      + | ENSG00000223972
DDX11L1
##   ENSG00000227232        1 14404-29570      - | ENSG00000227232
WASH7P
##   ENSG00000278267        1 17369-17436      - | ENSG00000278267
MIR6859-1
##   ENSG00000243485        1 29554-31109      + | ENSG00000243485
MIR1302-2
##   ENSG00000237613        1 34554-36081      - | ENSG00000237613
FAM138A
##   ENSG00000268020        1 52473-53312      + | ENSG00000268020
OR4G4P
##                            gene_biotype seq_coord_system      symbol
##                             <character>      <character> <character>
##   ENSG00000223972 transcribed_unproces..       chromosome     DDX11L1
##   ENSG00000227232 unprocessed_pseudogene       chromosome      WASH7P
##   ENSG00000278267                 miRNA       chromosome   MIR6859-1
##   ENSG00000243485               lincRNA       chromosome   MIR1302-2
##   ENSG00000237613               lincRNA       chromosome     FAM138A
##   ENSG00000268020 unprocessed_pseudogene       chromosome      OR4G4P
##                                       entrezid
##                                         <list>
##   ENSG00000223972 100287596,100287102,727856,...
##   ENSG00000227232                           <NA>
##   ENSG00000278267                      102466751
##   ENSG00000243485           105376912,100302278
##   ENSG00000237613          654835,645520,641702
##   ENSG00000268020                           <NA>
##   -------
##   seqinfo: 357 sequences (1 circular) from GRCh38 genome
```

Los datos se han extraído correctamente, así que ahora vamos a añadir la información de los genes a la tabla de expresión, asegurándonos que los nombres de los genes son los mismos en ambas tablas.

```
common_genes <- intersect(rownames(exprs_data), names(gene_ranges))
exprs_data <- exprs_data[common_genes, ]
gene_ranges <- gene_ranges[common_genes]
dim(exprs_data)

## [1] 57602   198
```

Verificamos que se ha hecho correctamente el filtrado de los genes y que ahora la tabla de expresión tiene 57.602 genes.

Vamos a actualizar el objeto `SummarizedExperiment` para que contenga la información de los genes, y que los genes sean equivalentes en todas las tablas

```
se <- SummarizedExperiment(
    assays = list(counts = as.matrix(exprs_data)),
    colData = metadata
)
rowRanges(se) <- gene_ranges
dim(se)
```

```
## [1] 57602   198
```

Se confirma finalmente que el objeto `SummarizedExperiment` tiene 57.602 genes y 198 muestras.

3. Selección de Muestras

En estas sección vamos a llevar a cabo la selección de las muestras que vamos a usar para el análisis. Vamos a filtrar las muestras para que contengan solo las muestras de los grupos COVID19, Bacterial y Healthy. Eliminamos también individuos duplicados, y convertimos las variables al formato adecuado. Además, vamos a seleccionar un conjunto de 75 muestras , utilizando una semilla aleatoria para garantizar la reproducibilidad del análisis.

Empezamos filtrando las muestras para que contengan solo los grupos de interés, y que no haya duplicados. También convertimos las variables al formato adecuado, y sustituimos caracteres como espacios, guiones, puntos, barras, etc., por guiones bajos para evitar problemas con los nombres de las variables.

```
table(metadata$cohort)
```

```
##
## Bacterial CoV other  COVID-19   healthy Influenza
##        24        61        77        19        17
```

```
library(dplyr)
```

```
##
## Adjuntando el paquete: 'dplyr'
```

```
## The following objects are masked from 'package:ensembldb':
##
##     filter, select
```

```
## The following object is masked from 'package:AnnotationDbi':
##
##     select
```

```
## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union
```

```
## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect

## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union

## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union

## The following object is masked from 'package:matrixStats':
##
##     count

## The following object is masked from 'package:Biobase':
##
##     combine

## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# Filtrar las muestras para que contengan solo los grupos de interés
filtered_metadata <- metadata %>%
    filter(cohort %in% c("COVID-19", "Bacterial", "healthy")) %>%
    distinct(rna_id, .keep_all = TRUE)
# Convertir las variables al formato adecuado
filtered_metadata$cohort <- as.factor(filtered_metadata$cohort)
filtered_metadata$gender <- as.factor(filtered_metadata$gender)
filtered_metadata$age <- as.numeric(filtered_metadata$age)
```

```
## Warning: NAs introducidos por coerción
```

```r
# Sustituir caracteres especiales por guiones bajos
filtered_metadata <- filtered_metadata %>%
    mutate(across(everything(), ~ gsub("[[:punct:]]", "_", .))) %>%
    mutate(across(everything(), ~ gsub(" ", "_", .))) %>%
    mutate(across(everything(), ~ gsub("/", "_", .))) %>%
    mutate(across(everything(), ~ gsub("\\\\", "_", .)))
colnames(exprs_data) <- colnames(exprs_data) %>%
    gsub("[[:punct:]]", "_", .) %>%
    gsub(" ", "_", .) %>%
```

```
    gsub("/", "_", .) %>%
    gsub("\\\\", "_", .)


# Verificar la estructura de los metadatos filtrados
head(filtered_metadata)
```

```
## # A tibble: 6 × 9
##   rna_id      subject_id age   gender race  cohort time_since_onset
hospitalized
##   <chr>       <chr>      <chr> <chr>  <chr> <chr>  <chr>
<chr>
## 1 94189       A1BD46     57    Female Blac… Bacte… <NA>
<NA>
## 2 DU18_02S00… 450905     60    Male   White COVID… early            No
## 3 DU18_02S00… 450905     60    Male   White COVID… early            No
## 4 DU18_02S00… 450905     60    Male   White COVID… early            No
## 5 DU18_02S00… 450905     60    Male   White COVID… middle           No
## 6 DU18_02S00… 450905     60    Male   White COVID… middle           No
## # ℹ 1 more variable: batch <chr>
```

```
dim(filtered_metadata)
```

```
## [1] 120   9
```

Como podemos ver, tras la selección y procesamiento, hemos obtenido la reducción a un total de 120 muestras.

Ahora vamos a seleccionar un subconjunto de 75 muestras aleatorias para el análisis. Para ello, vamos a usar la función `sample` de R, y vamos a establecer una semilla aleatoria para garantizar la reproducibilidad del análisis.

```
myseed <- sum(utf8ToInt("franciscojaviermorillajimenez"))
set.seed(myseed)

# Seleccionar un subconjunto de 75 muestras aleatorias
selected_samples <- sample(unique(filtered_metadata$rna_id), 75)
# Filtrar los metadatos para que contengan solo las muestras
seleccionadas
filtered_metadata <- filtered_metadata %>%
    filter(rna_id %in% selected_samples)
# Filtrar la matriz de expresión para que contenga solo las muestras
seleccionadas
filtered_exprs_data <- exprs_data[, colnames(exprs_data) %in%
filtered_metadata$rna_id]
# Verificar la estructura de los metadatos filtrados
head(filtered_metadata)
```

```
## # A tibble: 6 × 9
##   rna_id      subject_id age   gender race  cohort time_since_onset
hospitalized
```

```
##   <chr>       <chr>      <chr> <chr>  <chr> <chr>  <chr>
<chr>
## 1 94189       A1BD46     57    Female Blac… Bacte… <NA>
<NA>
## 2 DU18_02S00… 450905     60    Male   White COVID… early            No
## 3 DU18_02S00… 450905     60    Male   White COVID… early            No
## 4 DU18_02S00… 450905     60    Male   White COVID… middle           No
## 5 DU18_02S00… 450905     60    Male   White COVID… middle           No
## 6 DU18_02S00… 450905     60    Male   White COVID… late             No
## # ℹ 1 more variable: batch <chr>
```

```
dim(filtered_metadata)
```

```
## [1] 75  9
```

```
# Verificar la estructura de la matriz de expresión filtrada
head(filtered_exprs_data)
```

```
##                 94189 DU18_02S0011619 DU18_02S0011621 DU18_02S0011622
## ENSG00000223972     0               0              43               60
## ENSG00000227232   393             192             310              251
## ENSG00000278267     0               0              25               34
## ENSG00000243485     0               0               0                0
## ENSG00000237613     0               0               0                0
## ENSG00000268020     0               0               0                0
```

```
dim(filtered_exprs_data)
```

```
## [1] 57602    75
```

4. Preprocesado y Normalización

En esta sección, vamos a llevar a cabo el preprocesado y la normalización de los datos de expresión. Vamos a realizar la normalización mediante TPMS (Transcripts Per Million) y vamos a filtrar los genes con baja expresión.

Empezamos filtrando los genes con baja expresión. Es común usar un criterio de expresión mínima para filtrar los genes que tienen bajo conteo en la mayoría de las muestras. En este caso, vamos a usar un umbral de 10 conteos en al menos el 50% de las muestras.

```
# Filtrar los genes con baja expresión
filtered_exprs_data <- filtered_exprs_data[rowSums(filtered_exprs_data > 10) >= (0.5 * ncol(filtered_exprs_data)), ]
dim(filtered_exprs_data)
```

```
## [1] 16546    75
```

Así, hemos pasado de 57.602 genes a 16.000 genes, lo que es un número razonable para el análisis posterior.

vamos a actualizar el objeto `SummarizedExperiment` para que contenga solo los genes filtrados y las muestras seleccionadas.

```
# Actualizar el objeto SummarizedExperiment
se <- SummarizedExperiment(
    assays = list(counts = as.matrix(filtered_exprs_data)),
    colData = filtered_metadata
)
rowRanges(se) <- gene_ranges[gene_ranges$gene_id %in%
rownames(filtered_exprs_data)]
# Verificar
dim(se)
```

```
## [1] 16546    75
```

Filtrados los genes, vamos a aplicar la normalización mediante el método de TPM (Transcripts Per Million). Este método normaliza los datos de expresión teniendo en cuenta la longitud de los genes y el número total de lecturas en cada muestra. La fórmula que se utiliza para calcular los TPM es la siguiente:

$$TPM = \frac{(conteo\ de\ lecturas\ del\ gen)/(longitud\ del\ gen)}{(suma\ de\ todos\ los\ conteos\ de\ lecturas)/(longitud\ total)} * 10^6$$

Donde la longitud del gen se expresa en kilobases (kb) y el número total de lecturas se expresa en millones.

Para aplicar la normalización TPM, vamos a calcular la longitud de los genes y luego aplicar la fórmula anterior. Vamos a usar el paquete `edgeR` para calcular la longitud de los genes y normalizar los datos.

```
library(edgeR)
```

```
## Warning: package 'edgeR' was built under R version 4.4.2
```

```
## Cargando paquete requerido: limma
```

```
## Warning: package 'limma' was built under R version 4.4.2
```

```
##
## Adjuntando el paquete: 'limma'
```

```
## The following object is masked from 'package:BiocGenerics':
##
##     plotMA
```

```
# Calcular la longitud de los genes
gene_lengths <- width(rowRanges(se))
rpk <- filtered_exprs_data / (gene_lengths / 1000)  # Dividir por
```

```
longitud en kb
scaling_factor <- colSums(rpk) / 1e6  # Escalar por millón de RPKs
tpm <- sweep(rpk, 2, scaling_factor, FUN = "/")  # Dividir cada columna
por el factor de escal
assays(se, withDimnames=FALSE)[["TPM"]] <- tpm
```

Confirmamos que se ha añadido la matriz de TPM al objeto `SummarizedExperiment` y que tiene las dimensiones correctas.

```
# Verificar la matriz de TPM
assayNames(se)

## [1] "counts" "TPM"

assay(se, "TPM")[1:10, 1:5]

##                        94189 DU18_02S0011619 DU18_02S0011621
DU18_02S0011622
## ENSG00000227232    3.1428661       2.4171887       2.9425808
2.0032542
## ENSG00000278267    0.0000000       0.0000000      52.9294627
60.5246158
## ENSG00000238009    0.4613741       0.4383718       0.2009054
0.2997010
## ENSG00000268903  747.0316690     411.9866511     911.0990277
1053.3688101
## ENSG00000269981 2424.9906038    2419.7612544    2176.7577002
2443.5748056
## ENSG00000239906   19.8478209       0.0000000      48.4256466
77.2514187
## ENSG00000241860    0.3033336       0.4716300       0.1466840
0.2466655
## ENSG00000279928    0.0000000       0.1081231       3.7500195
0.0000000
## ENSG00000279457    5.6708060       3.5461273       8.7409227
4.8576899
## ENSG00000228463    0.0000000       0.5491502       0.2796626
1.3253454
##                 DU18_02S0011626
## ENSG00000227232       0.7074890
## ENSG00000278267     189.0146940
## ENSG00000238009       0.2388658
## ENSG00000268903     340.9453459
## ENSG00000269981    1006.8653263
## ENSG00000239906       0.0000000
## ENSG00000241860       0.3276596
## ENSG00000279928       6.3432363
## ENSG00000279457       4.4563598
## ENSG00000228463       0.0000000

assay(se, "counts")[1:10, 1:5]
```

```
##                  94189 DU18_02S0011619 DU18_02S0011621 DU18_02S0011622
## ENSG00000227232   393              192              310              251
## ENSG00000278267     0                0               25               34
## ENSG00000238009   169              102               62              110
## ENSG00000268903  4650             1629             4778             6570
## ENSG00000269981  5678             3599             4294             5733
## ENSG00000239906    90                0              185              351
## ENSG00000241860    81               80               33               66
## ENSG00000279928     0                1               46                0
## ENSG00000279457   720              286              935              618
## ENSG00000228463     0              114               77              434
##                  DU18_02S0011626
## ENSG00000227232               91
## ENSG00000278267              109
## ENSG00000238009               90
## ENSG00000268903             2183
## ENSG00000269981             2425
## ENSG00000239906                0
## ENSG00000241860               90
## ENSG00000279928               95
## ENSG00000279457              582
## ENSG00000228463                0
```

```r
dim(assay(se, "TPM"))
```

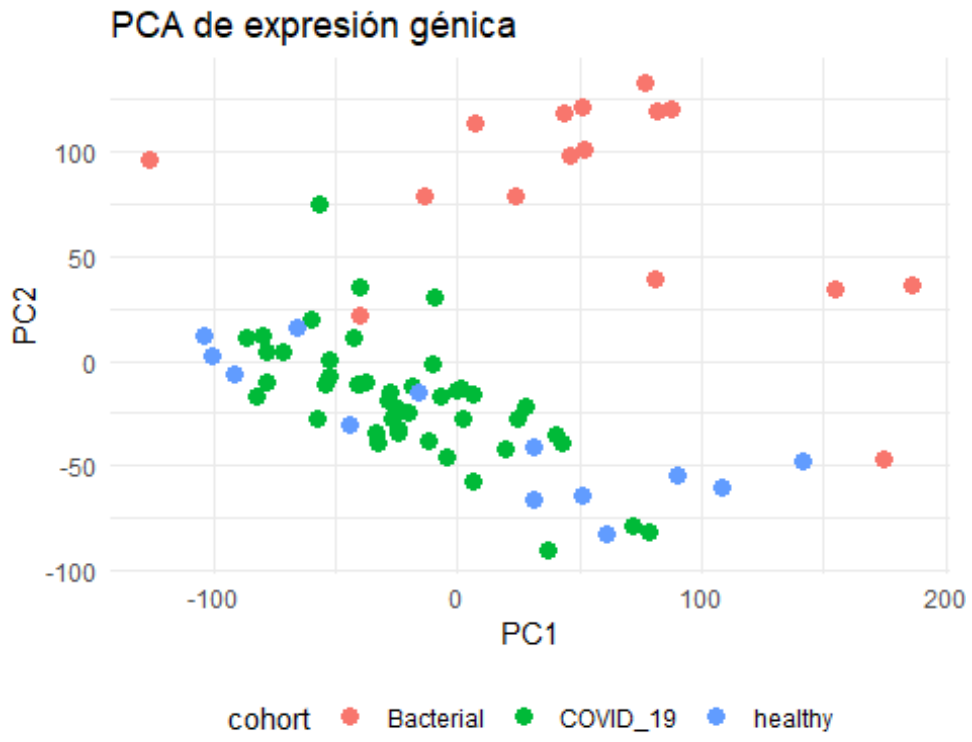
```
## [1] 16546    75
```

5. Análisis Exploratorio de Datos

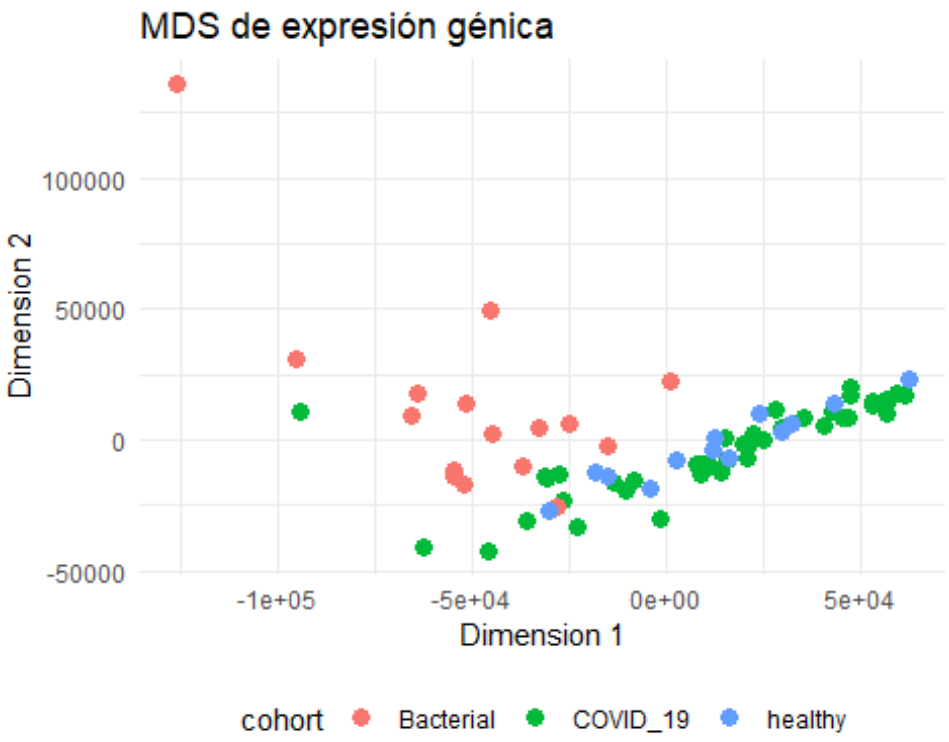
En esta sección, vamos a llevar a cabo un análisis exploratorio de los datos. Vamos a realizar un análisis de componentes principales (PCA) y MDS para visualizar la variabilidad de los datos y la relación entre las muestras. Además, vamos a llevar a cabo un análisis de agrupamiento jerárquico (HCA) para visualizar la variabilidad de los datos y la relación entre las muestras, y un mapa de calor. Como objetivo final de esta sección, vamos a identificar las variables confusoras que pueden influir en el análisis de expresión diferencial.

```r
library(ggplot2)
library(SummarizedExperiment)

# 1. Extraer matriz de expresión TPM y metadatos filtrados
tpm_mat <- assay(se, "TPM")
meta <- colData(se)

# 2. PCA
pca <- prcomp(t(tpm_mat), center = TRUE, scale. = TRUE)
pca_df <- as.data.frame(pca$x)
pca_df$sample <- rownames(pca_df)
pca_df$cohort <- meta$cohort[match(pca_df$sample, rownames(meta))]
```

```r
pca_df$gender <- meta$gender[match(pca_df$sample, rownames(meta))]
pca_df$age <- meta$age[match(pca_df$sample, rownames(meta))]

# 3. Gráfico PCA
ggplot(pca_df, aes(x = PC1, y = PC2, color = cohort)) +
  geom_point(size = 3) +
  labs(title = "PCA de expresión génica", x = "PC1", y = "PC2") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



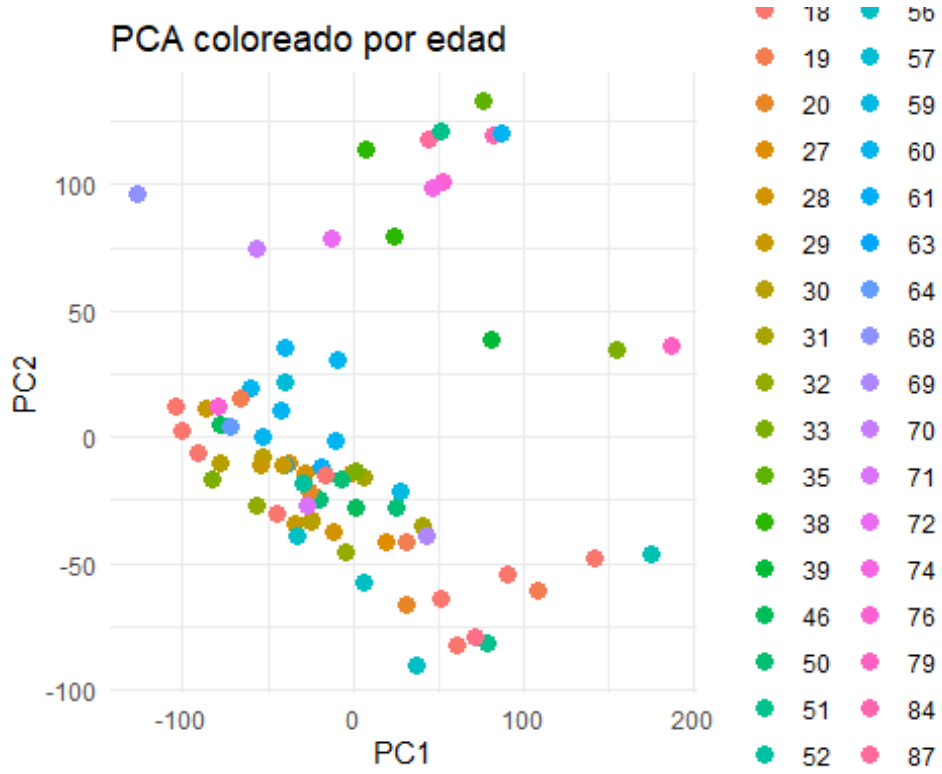

```r
# 4. MDS
mds <- cmdscale(dist(t(tpm_mat)), k = 2)
mds_df <- as.data.frame(mds)
mds_df$sample <- rownames(mds_df)
mds_df$cohort <- meta$cohort[match(mds_df$sample, rownames(meta))]
mds_df$gender <- meta$gender[match(mds_df$sample, rownames(meta))]
mds_df$age <- meta$age[match(mds_df$sample, rownames(meta))]

# 5. Gráfico MDS
ggplot(mds_df, aes(x = V1, y = V2, color = cohort)) +
  geom_point(size = 3) +
  labs(title = "MDS de expresión génica", x = "Dimension 1", y =
"Dimension 2") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



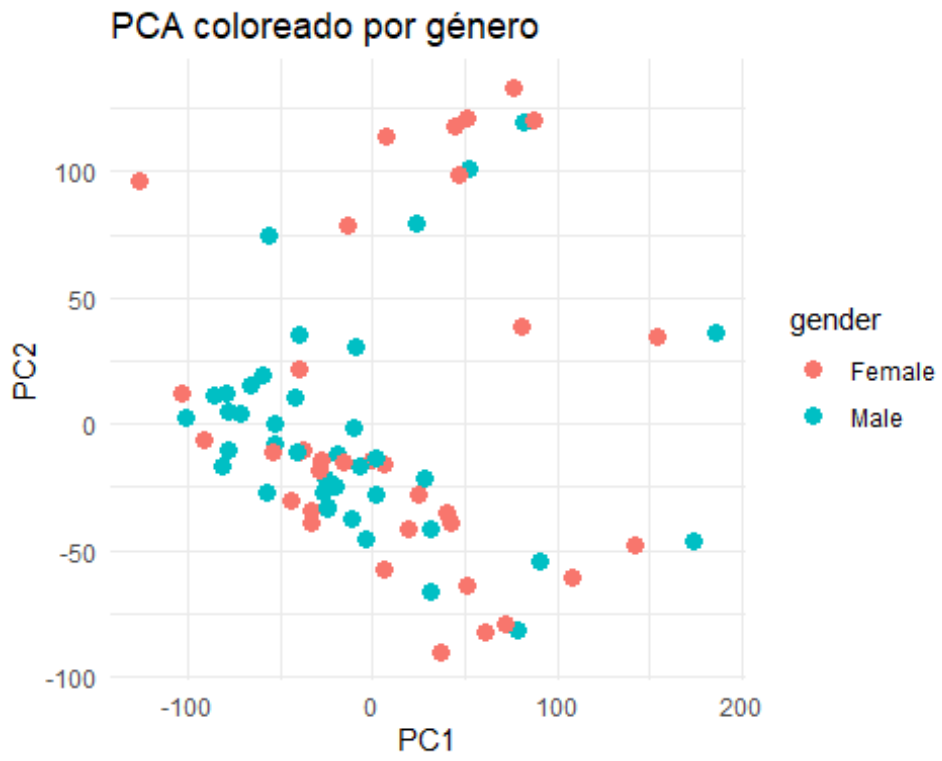
La visualización del PCA por cohort nos permite evaluar si las muestras se agrupan según las características biológicas. Si observamos una separación clara, podemos inferir que la expresión génica está influenciada por el grupo de estudio. Tanto el análisis de componentes principales (PCA) como el escalado multidimensional (MDS) sugieren que existe una separación entre las tres cohortes (Bacterial, COVID_19 y healthy) en los datos de expresión génica. En ambas visualizaciones, las muestras bacterianas tienden a agruparse de manera diferenciada, mientras que las muestras COVID_19 y healthy aparecen algo más próximas entre sí, aunque siguen mostrando cierta distinción. Estos resultados podrían indicar diferencias en los perfiles de expresión génica entre los grupos.

Vamos a verificar si hay otras variables que pueden influir en el análisis de expresión diferencial. Para ello, hacemos el PCA con las variables gender y age para ver si hay alguna separación clara entre las muestras.

```
ggplot(pca_df, aes(x = PC1, y = PC2, color = age)) +
  geom_point(size = 3) +
  labs(title = "PCA coloreado por edad") +
  theme_minimal()
```




```r
ggplot(pca_df, aes(x = PC1, y = PC2, color = gender)) +
  geom_point(size = 3) +
  labs(title = "PCA coloreado por género") +
  theme_minimal()
```



Ninguna de las dos variables parece influir en el análisis de expresión diferencial, ya que no hay una separación clara entre las muestras. Vamos a ver cuales son la sdemas variables disponibles en los metadatos.

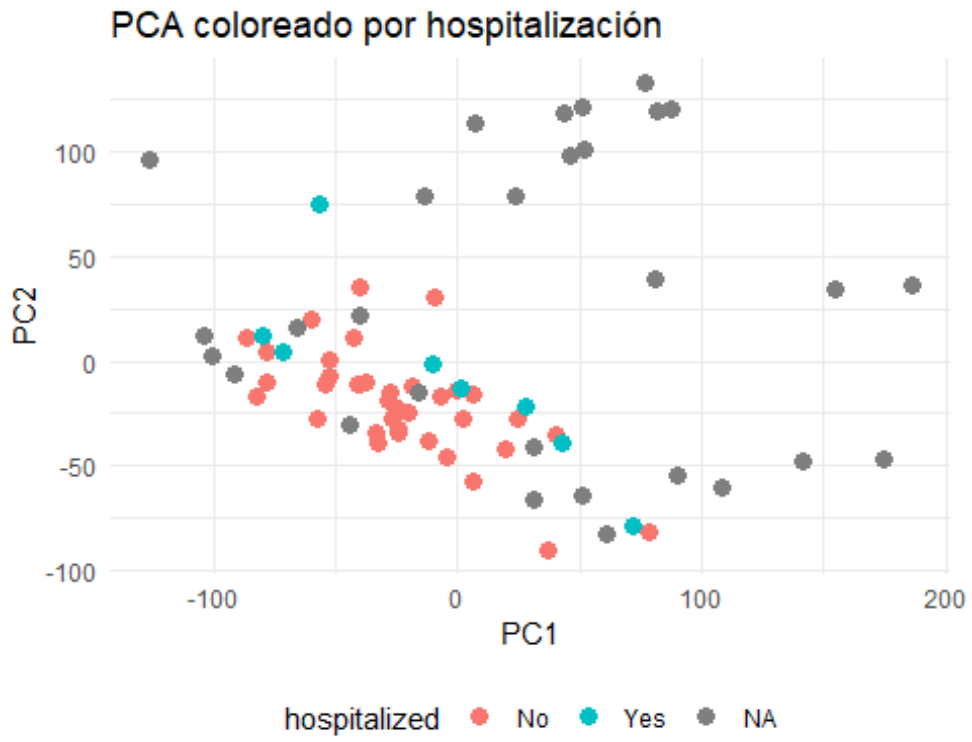
```
colnames(filtered_metadata)
```

```
## [1] "rna_id"           "subject_id"       "age"              "gender"
## [5] "race"             "cohort"           "time_since_onset"
"hospitalized"
## [9] "batch"
```

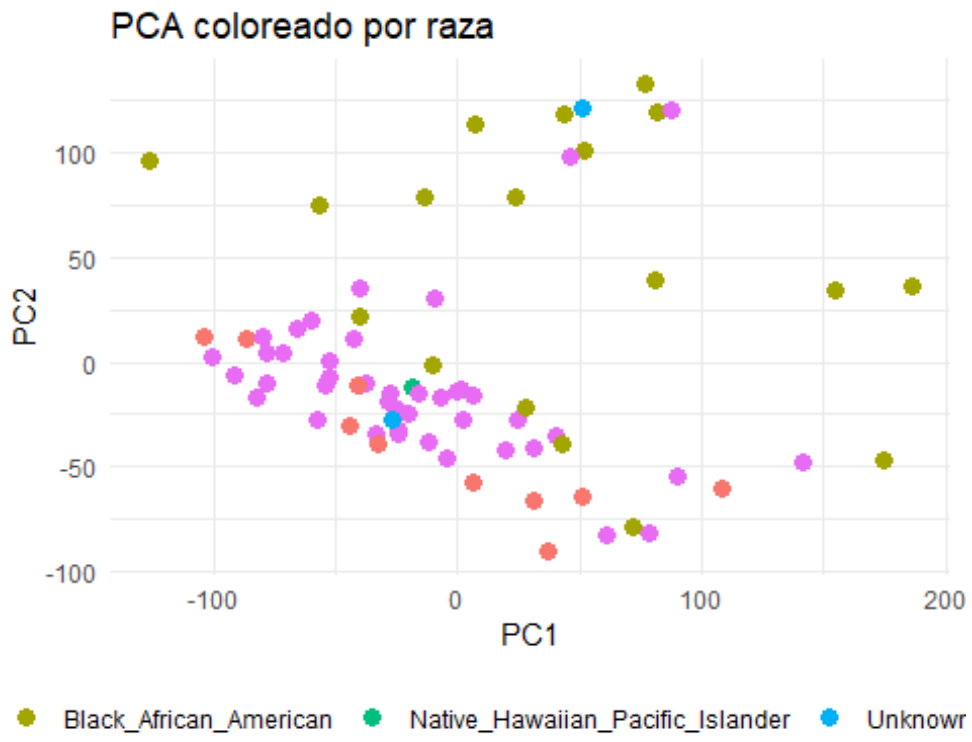
Vemos que hay una variable `cohort` que es la que hemos usado para el análisis, y las variables `gender` y `age` que ya hemos analizado. Además, hay otras variables como `hospitalized`, `batch`, `race`, etc. Vamos a ver si alguna de estas variables puede influir en el análisis de expresión diferencial.

```
tpm_mat <- assay(se, "TPM")
meta <- colData(se)

# Realizar PCA
pca <- prcomp(t(tpm_mat), center = TRUE, scale. = TRUE)
pca_df <- as.data.frame(pca$x)
pca_df$sample <- rownames(pca_df)
pca_df$hospitalized <- meta$hospitalized[match(pca_df$sample,
rownames(meta))]
pca_df$race <- meta$race[match(pca_df$sample, rownames(meta))]
pca_df$batch <- meta$batch[match(pca_df$sample, rownames(meta))]
```

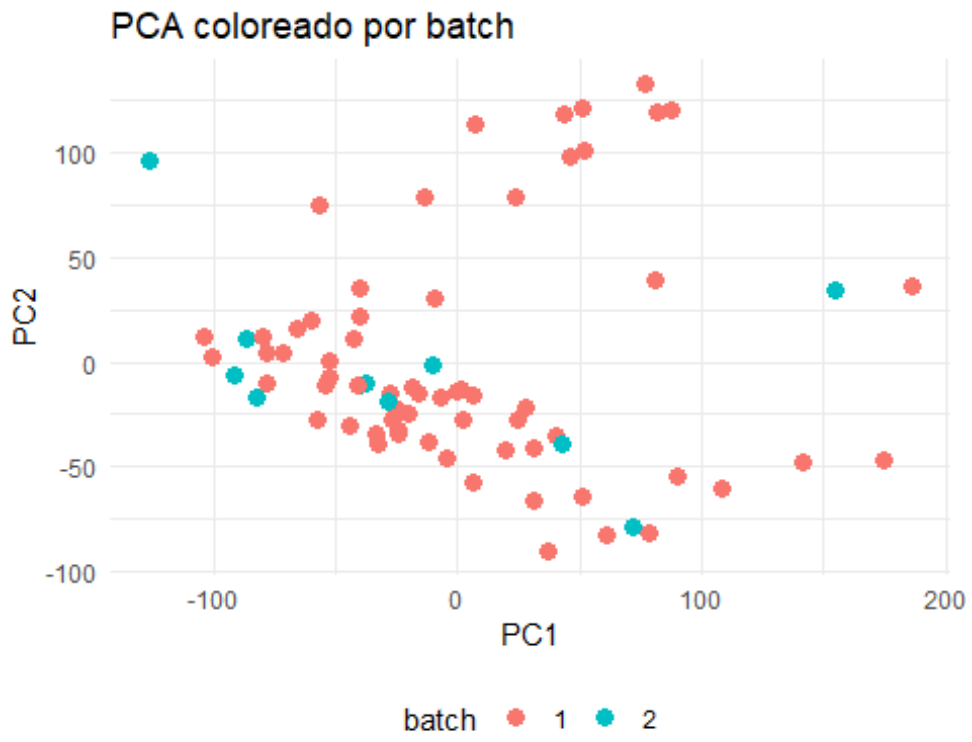
```r
# Visualización coloreada por hospitalización
ggplot(pca_df, aes(x = PC1, y = PC2, color = hospitalized)) +
  geom_point(size = 3) +
  labs(title = "PCA coloreado por hospitalización", x = "PC1", y = "PC2")
+
  theme_minimal() +
  theme(legend.position = "bottom")
```




```r
# Visualización coloreada por raza
ggplot(pca_df, aes(x = PC1, y = PC2, color = race)) +
  geom_point(size = 3) +
  labs(title = "PCA coloreado por raza", x = "PC1", y = "PC2") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



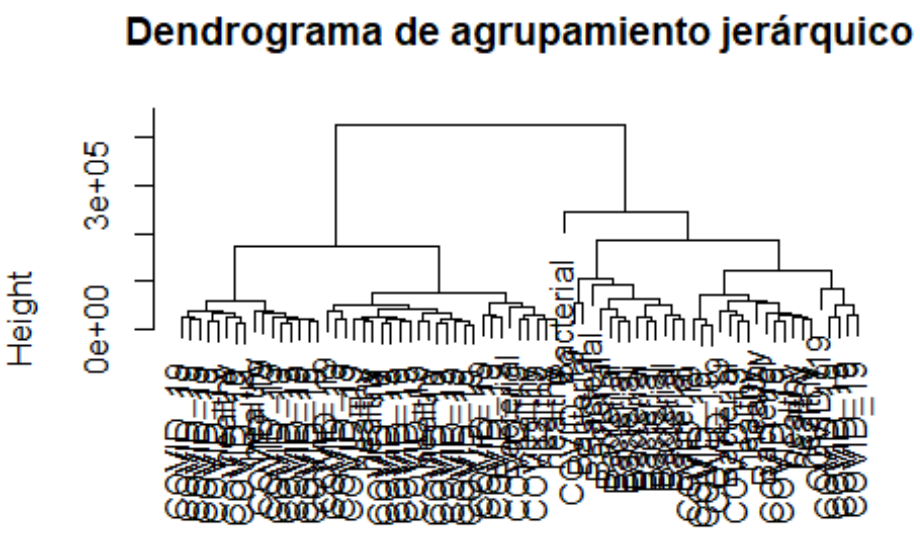

```r
# Visualización coloreada por batch (lote experimental)
ggplot(pca_df, aes(x = PC1, y = PC2, color = batch)) +
  geom_point(size = 3) +
  labs(title = "PCA coloreado por batch", x = "PC1", y = "PC2") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



Tampoco parecen influir de manera significativa en el análisis, así que no se toman en cuenta para el análisis de expresión diferencial.

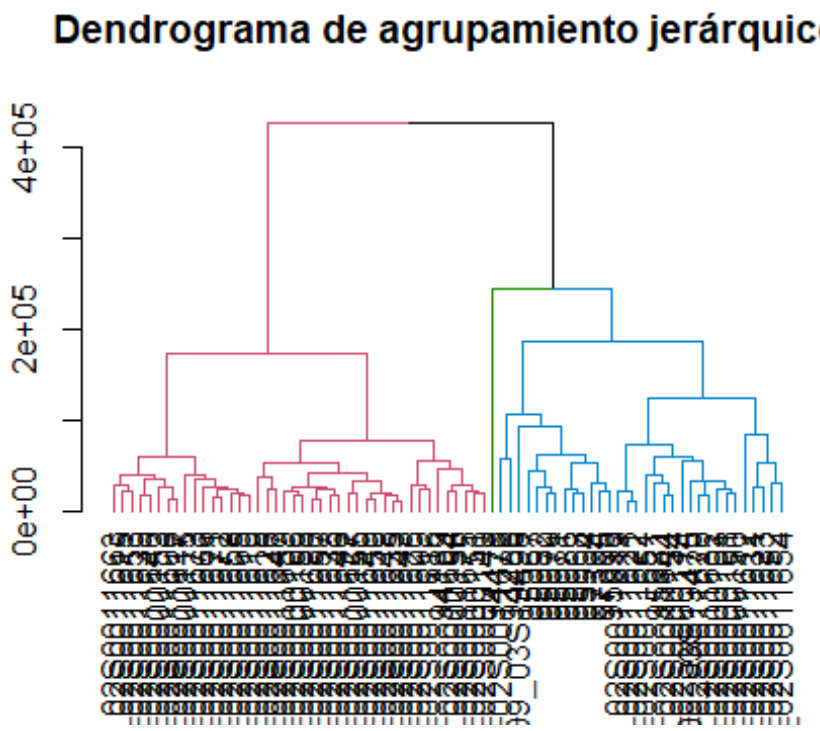
Ahora vamos a realizar un análisis de agrupamiento jerárquico (HCA) para visualizar la agrupación entre los datos. Vamos a usar la función `hclust` de R para realizar el análisis de agrupamiento jerárquico, y vamos a graficar el dendrograma resultante.

```
# Calcular la matriz de distancias
dist_matrix <- dist(t(assay(se, "TPM")), method = "euclidean")
# Realizar el análisis de agrupamiento jerárquico
hclust_result <- hclust(dist_matrix, method = "ward.D2")
# Graficar el dendrograma
plot(hclust_result, labels = filtered_metadata$cohort, main =
"Dendrograma de agrupamiento jerárquico", xlab = "", sub = "")
```




```
# Graficar el dendrograma con colores
library(dendextend)

## Warning: package 'dendextend' was built under R version 4.4.3

##
## ---------------------
## Welcome to dendextend version 1.19.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at:
https://github.com/talgalili/dendextend/issues
## You may ask questions at stackoverflow, use the r and dendextend tags:
##   https://stackoverflow.com/questions/tagged/dendextend
##
##  To suppress this message use:
suppressPackageStartupMessages(library(dendextend))
## ---------------------

##
## Adjuntando el paquete: 'dendextend'

## The following object is masked from 'package:stats':
##
##     cutree
```

```
dend <- as.dendrogram(hclust_result)
dend <- color_branches(dend, k = 3)

## Loading required namespace: colorspace

plot(dend, main = "Dendrograma de agrupamiento jerárquico", xlab = "",
sub = "")
```



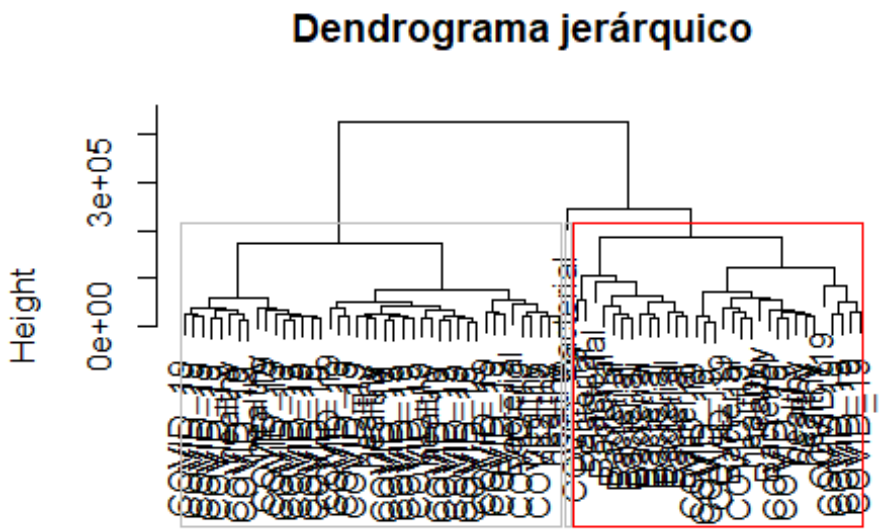
En el primer clustering jerárquico, se ha detectado 1 muestra que podría separarse del resto. Por ello, en el culstering de colores, se selecciona un valor de k de 3 para ver si se separa esta. Efectivamente, vemos como la muestra se separa del resto. Vamos a identificar qué muestra es la que se separa del resto. Para ello, vamos a obtener las alturas de los nodos del clustering y a extraer los nodos que superan un umbral alto.

```
heights <- hclust_result$height
threshold <- quantile(heights, 0.99)
outlier_indices <- which(heights > threshold)
print(data.frame(merge = outlier_indices, height =
heights[outlier_indices]))

##   merge   height
## 1    74 426321.6

k <- 3
clust <- cutree(hclust_result, k = k)
cluster_sizes <- table(clust)
outlier_cluster <-
as.numeric(names(cluster_sizes)[which.min(cluster_sizes)])
```

```
outlier_samples <- names(clust)[clust == outlier_cluster]
print(outlier_samples)
```

```
## [1] "94478"
```

```
plot(hclust_result, labels = filtered_metadata$cohort, main =
"Dendrograma jerárquico", xlab = "", sub = "")
rect.hclust(hclust_result, k = k, border = c("grey", "grey", "red")[1:k])
```



Vemos que la muestra que se separa del resto es la muestra “94478”. vamos a extraer la información de esta muestra de los metadatos para ver qué información tenemos de ella.

```
outlier_sample <- filtered_metadata[filtered_metadata$rna_id ==
outlier_samples, ]
print(outlier_sample)
```

```
## # A tibble: 1 × 9
##   rna_id subject_id age   gender race       cohort time_since_onset
hospitalized
##   <chr>  <chr>      <chr> <chr>  <chr>      <chr>  <chr>
<chr>
## 1 94478  896282     68    Female Black_Afr… Bacte… <NA>
<NA>
## # ℹ 1 more variable: batch <chr>
```

Se corresponde con una muestra Bacterial de una mujer de 68 años. Vamos a eliminar esta muestra del objeto SE.

```r
se <- se[, !colnames(se) %in% outlier_samples]
dim(se)
```

```
## [1] 16546    74
```

Se confirma que se ha eliminado la muestra del objeto `SummarizedExperiment`, ya que ahora tiene 74 muestras en lugar de 75.

6. Análisis de Expresión Diferencial

En esta sección, vamos a llevar a cabo el análisis de expresión diferencial. Para llevar a cabo un análisis de expresión diferencial de RNA-seq en R, existen varios enfoques, entre los que destacan `edgeR`, `DESeq2` y `limma`. La diferencia entre ellos radica principalmente en la forma en que modelan los datos y en los métodos de normalización que utilizan. Para seleccionar el método de análisis, vamos a usar una semilla aleatoria para seleccionar uno de los métodos disponibles.

```r
set.seed(myseed)
sample(c("edgeR", "voom+limma", "DESeq2"), size = 1)
```

```
## [1] "edgeR"
```

En este caso ha tocado usar el método `edgeR`, que es un método de análisis de expresión diferencial basado en la teoría de modelos lineales generalizados. Este método es adecuado para datos de RNA-seq y tiene en cuenta la variabilidad biológica y técnica en los datos.

Para empezar, vamos a construir la matriz de diseño y las matrices de contrastes adecuadas para evaluar la expresión génica diferencial en las comparaciones Bacterial vs healthy y COVID19 vs healthy. Como hemos visto en el análisis exploratorio de datos, la variable `cohort` es la que nos interesa, y vamos a usarla como variable de interés. Además, vamos a incluir las variables `gender` y `age` como variables confusoras en el análisis.

```r
media_edad <- mean(colData(se)$age, na.rm = TRUE)
```

```
## Warning in mean.default(colData(se)$age, na.rm = TRUE): argument is
not numeric
## or logical: returning NA
```

```r
media_edad
```

```
## [1] NA
```

Calculamos la media de edad para asignarla a una muestra que tiene el valor de edad faltante. Creamos ya la matriz de diseño y la matriz de contrastes.

```r
library(edgeR)
library(limma)
```

```r
# Definir healthy como el grupo de referencia
colData(se)$age[is.na(colData(se)$age)] <- 41
colData(se)$cohort <- relevel(factor(colData(se)$cohort), ref =
"healthy")
colData(se)$age <- as.numeric(as.character(colData(se)$age))
design <- model.matrix(~ cohort + gender + age, data = colData(se))

head(design)
```

```
##                 (Intercept) cohortBacterial cohortCOVID_19 genderMale
age
## 94189                     1               1              0          0
57
## DU18_02S0011619           1               0              1          1
60
## DU18_02S0011621           1               0              1          1
60
## DU18_02S0011622           1               0              1          1
60
## DU18_02S0011626           1               0              1          1
60
## DU18_02S0011625           1               0              1          1
60
```

```r
# Definir los contrastes
contrast_matrix <- makeContrasts(
  Bacterial_vs_Healthy = cohortBacterial,
  COVID19_vs_Healthy = cohortCOVID_19,
  levels = design
)
```

```
## Warning in makeContrasts(Bacterial_vs_Healthy = cohortBacterial,
## COVID19_vs_Healthy = cohortCOVID_19, : Renaming (Intercept) to
Intercept
```

```r
contrast_matrix
```

```
##                  Contrasts
## Levels            Bacterial_vs_Healthy COVID19_vs_Healthy
##   Intercept                          0                  0
##   cohortBacterial                    1                  0
##   cohortCOVID_19                     0                  1
##   genderMale                         0                  0
##   age                                0                  0
```

Ya que tenemos el diseño y los contrastes definidos, vamos a realizar el análisis de expresión diferencial utilizando el método `edgeR`. Como se vio, tenemos los datos normalizados y con las muestras seleccionadas de manera aleatoria en el assay "TPM". Vamos a crear el objeto `DGEList` a partir del objeto `SummarizedExperiment` y a realizar el análisis de expresión diferencial.

Para seleccionar los genes diferencialmente expresados, vamos a usar un umbral de significación estadística de 0.05 y un umbral de log2FC de 1.5. El umbral del valor de 0.05 de FDR selecciona genes con significancia desde el punto de vista estadístico, mientras que el log2FC de 1.5 selecciona genes con significancia biológica.

```
# Crear el objeto DGEList
dge <- DGEList(counts = assay(se, "counts"), group = colData(se)$cohort)
# Normalizar los datos
dge <- calcNormFactors(dge)
# Ajustar el modelo
dge <- estimateDisp(dge, design)
fit <- glmQLFit(dge, design)
# Realizar el análisis de expresión diferencial
qlf <- glmQLFTest(fit, contrast = contrast_matrix)
# Obtener los resultados
results <- topTags(qlf, n = Inf)
head(results$table)
```

```
##                 logFC.Bacterial_vs_Healthy logFC.COVID19_vs_Healthy
logCPM
## ENSG00000123836                   5.520477              -0.55018539
6.541613
## ENSG00000079215                   5.725905               0.41534298
3.262725
## ENSG00000236525                   4.205697              -0.34796414
2.050104
## ENSG00000090376                   3.513372              -0.60434749
7.534944
## ENSG00000119402                   1.858089              -0.14377255
6.295405
## ENSG00000150403                   2.563367              -0.02906945
5.343273
##                         F       PValue          FDR
## ENSG00000123836 162.9917 2.997743e-28 4.960066e-24
## ENSG00000079215 114.5476 1.629565e-27 1.348139e-23
## ENSG00000236525 112.0759 1.258542e-26 6.941278e-23
## ENSG00000090376 141.3319 1.848250e-26 7.645286e-23
## ENSG00000119402 139.6935 2.624433e-26 7.953305e-23
## ENSG00000150403 139.3191 2.884070e-26 7.953305e-23
```

Ahora que tenemos los resultados del análisis de expresión diferencial, vamos a filtrar los genes diferencialmente expresados utilizando el umbral de significación estadística y el umbral de log2FC. Vamos a usar un umbral de 0.05 para el valor de FDR y un umbral de 1.5 para el log2FC.

```
# Filtrar los genes diferencialmente expresados
results_filtered_BACTERIAL <- results$table[results$table$FDR < 0.05 &
abs(results$table$logFC.Bacterial_vs_Healthy
) > 1.5, ]
```

```r
results_filtered_COVID19 <- results$table[results$table$FDR < 0.05 &
abs(results$table$logFC.COVID19_vs_Healthy
) > 1.5, ]

head(results_filtered_BACTERIAL)
```

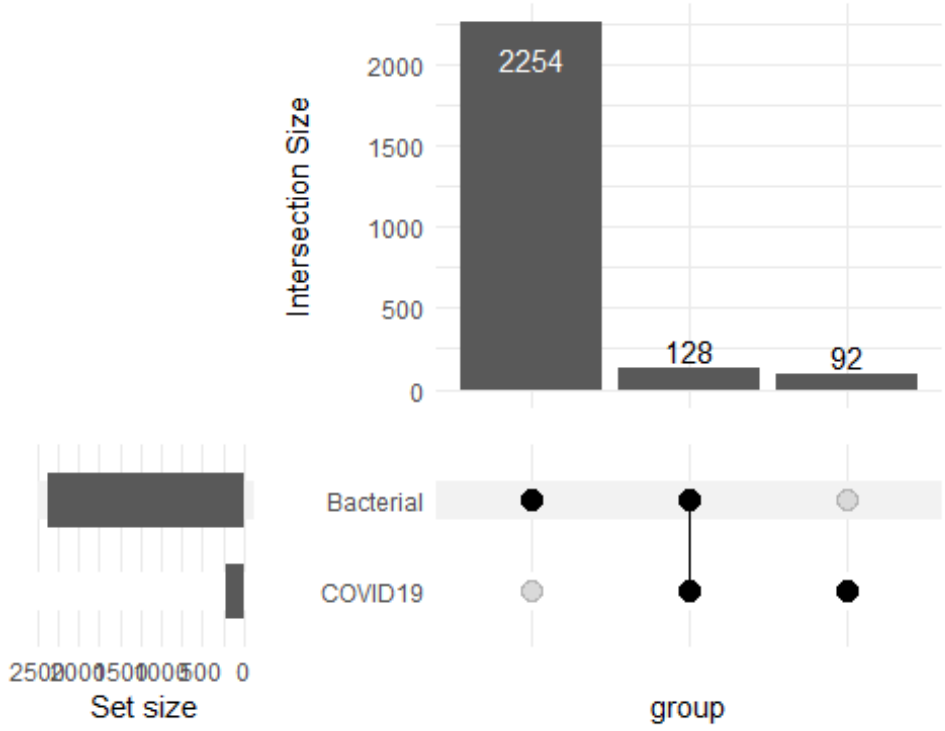
```
##                 logFC.Bacterial_vs_Healthy logFC.COVID19_vs_Healthy
logCPM
## ENSG00000123836                   5.520477              -0.55018539
6.541613
## ENSG00000079215                   5.725905               0.41534298
3.262725
## ENSG00000236525                   4.205697              -0.34796414
2.050104
## ENSG00000090376                   3.513372              -0.60434749
7.534944
## ENSG00000119402                   1.858089              -0.14377255
6.295405
## ENSG00000150403                   2.563367              -0.02906945
5.343273
##                         F       PValue          FDR
## ENSG00000123836 162.9917 2.997743e-28 4.960066e-24
## ENSG00000079215 114.5476 1.629565e-27 1.348139e-23
## ENSG00000236525 112.0759 1.258542e-26 6.941278e-23
## ENSG00000090376 141.3319 1.848250e-26 7.645286e-23
## ENSG00000119402 139.6935 2.624433e-26 7.953305e-23
## ENSG00000150403 139.3191 2.884070e-26 7.953305e-23
```

```r
head(results_filtered_COVID19)
```

```
##                 logFC.Bacterial_vs_Healthy logFC.COVID19_vs_Healthy
logCPM
## ENSG00000259379                 1.94904700               -2.132790
1.1222749
## ENSG00000202198                -0.07594784               -2.969101
7.0404288
## ENSG00000279166                 1.81288421               -1.635755
0.5264003
## ENSG00000196074                 1.87497056               -1.503179
1.9587294
## ENSG00000166527                 2.03375761               -2.099800
4.4291066
## ENSG00000173578                 5.27336885                1.865257
0.8417729
##                         F       PValue          FDR
## ENSG00000259379 59.65382 2.802444e-18 2.743742e-16
## ENSG00000202198 70.18478 4.655004e-18 4.255343e-16
## ENSG00000279166 54.97381 2.134063e-17 1.471258e-15
## ENSG00000196074 63.33560 3.689348e-17 2.347844e-15
## ENSG00000166527 64.07100 5.021230e-17 3.021137e-15
## ENSG00000173578 52.68928 9.559276e-17 5.307644e-15
```

Para terminar el análisis de expresión diferencial, vamos a comparar los resultados de ambos contrastes usando Upset plots, que son una forma de visualizar la intersección entre conjuntos de datos. Vamos a usar el paquete `ComplexUpset` para crear los gráficos, usando la lista de genes diferencialmente expresados de cada contraste, creando un nuevo dataframe con la presencia/ausencia de genes en cada contraste, y usándolo para crear el gráfico.

```
library(ComplexUpset)

## Warning: package 'ComplexUpset' was built under R version 4.4.3

library(dplyr)
library(tidyr)

##
## Adjuntando el paquete: 'tidyr'

## The following object is masked from 'package:S4Vectors':
##
##     expand

library(ggplot2)

# Crear listas de genes diferencialmente expresados
genes_Bacterial <- rownames(results_filtered_BACTERIAL)
genes_COVID19 <- rownames(results_filtered_COVID19)

# Crear un dataframe con presencia/ausencia de genes
upset_data <- data.frame(
  Gene = unique(c(genes_Bacterial, genes_COVID19)),  # Lista única de
genes
  Bacterial = as.integer(unique(c(genes_Bacterial, genes_COVID19)) %in%
genes_Bacterial),
  COVID19 = as.integer(unique(c(genes_Bacterial, genes_COVID19)) %in%
genes_COVID19)
)

# Crear el gráfico UpSet
upset(
  upset_data,
  intersect = c("Bacterial", "COVID19"),  # Cambiar `sets=` por
`intersect=`
  mode = "distinct",  # Definir el modo correcto
  base_annotations = list(
    'Intersection Size' = intersection_size()
  )
)
```



Gracias al gráfico UpSet, podemos ver que hay un número significativo de genes que son diferencialmente expresados en ambos contrastes llegando a 150 genes comunes, lo que sugiere que hay una superposición en los perfiles de expresión génica entre las muestras Bacterial y COVID19. Esto podría indicar que hay mecanismos biológicos comunes de respuesta a infección en ambos grupos.

7. Análisis de sobrerepresentación

Por último, llevamos a cabo el análisis de sobrerepresentación para identificar las funciones enriquecidas entre los genes sobreexpresados en pacientes con COVID19 en comparación con los controles sanos. Para ello, vamos a usar el paquete `clusterProfiler` y la base de datos `org.Hs.eg.db` para realizar el análisis de enriquecimiento de GO. Vamos a usar la función `enrichGO` para realizar el análisis y luego vamos a filtrar los resultados para obtener solo los términos significativos.

En este análisis se busca obtener información sobre los procesos biológicos que están alterados en los pacientes con COVID19 en comparación con los controles sanos. Para ello, se seleccionan los genes diferencialmente expresados en el contraste COVID19 vs healthy, y se usa como background la lista de genes presentes en la matriz de expresión de la cohorte healthy.

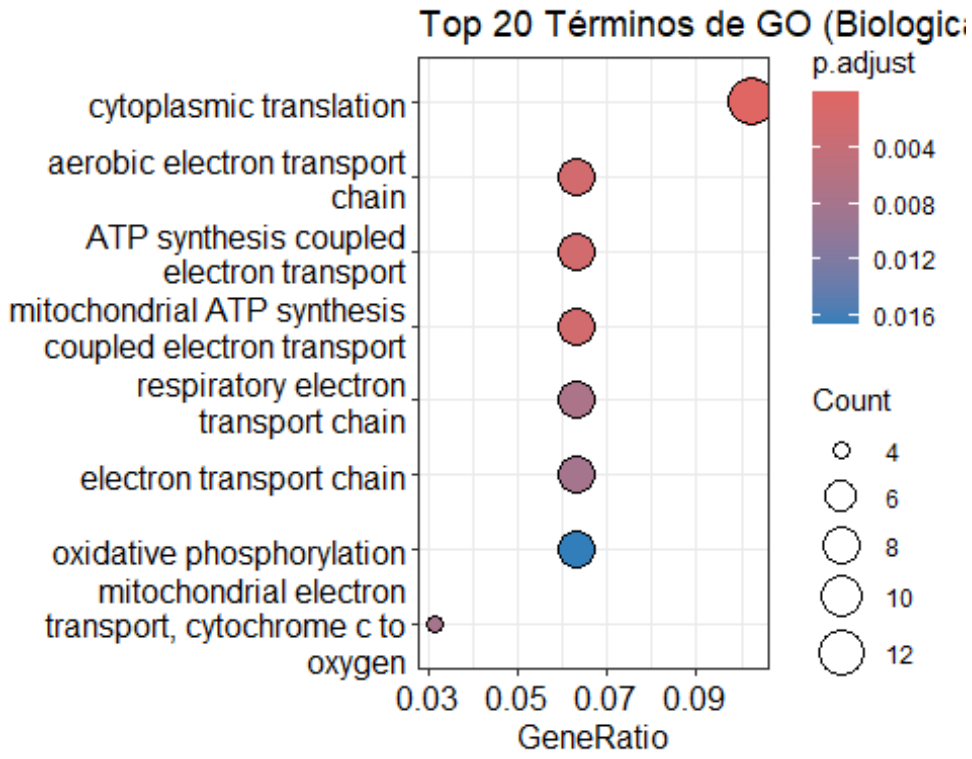
```
library(clusterProfiler)

## Warning: package 'clusterProfiler' was built under R version 4.4.2
```

```
##

## clusterProfiler v4.14.6 Learn more at https://yulab-
smu.top/contribution-knowledge-mining/
##
## Please cite:
##
## Guangchuang Yu, Li-Gen Wang, Yanyan Han and Qing-Yu He.
## clusterProfiler: an R package for comparing biological themes among
## gene clusters. OMICS: A Journal of Integrative Biology. 2012,
## 16(5):284-287

##
## Adjuntando el paquete: 'clusterProfiler'

## The following objects are masked from 'package:ensembldb':
##
##     filter, select

## The following object is masked from 'package:AnnotationDbi':
##
##     select

## The following object is masked from 'package:IRanges':
##
##     slice

## The following object is masked from 'package:S4Vectors':
##
##     rename

## The following object is masked from 'package:stats':
##
##     filter

library(org.Hs.eg.db)

##

# Seleccionar muestras de la cohorte Healthy y extraer la matriz de
expresión
samples_healthy <- colnames(se)[colData(se)$cohort == "healthy"]
tpm_healthy <- assay(se, "TPM")[, samples_healthy]

# Convertir los nombres de los genes a IDs de Entrez
gene_ids <- rownames(tpm_healthy)
entrez_ids <- mapIds(org.Hs.eg.db, keys = gene_ids, column = "ENTREZID",
keytype = "ENSEMBL", multiVals = "first")

## 'select()' returned 1:many mapping between keys and columns
```

```r
gene_ids_covid19 <- mapIds(org.Hs.eg.db, keys =
rownames(results_filtered_COVID19), column = "ENTREZID", keytype =
"ENSEMBL", multiVals = "first")

## 'select()' returned 1:many mapping between keys and columns

# Realizar el análisis de enriquecimiento de GO
go_results <- enrichGO(
  gene = gene_ids_covid19,
  universe = entrez_ids,
  OrgDb = org.Hs.eg.db,
  keyType = "ENTREZID",
  ont = "BP",  # Usar solo el dominio de Biológico
  pAdjustMethod = "BH",
  qvalueCutoff = 0.05,
  readable = TRUE
)

# Filtrar los resultados para obtener solo los términos significativos
go_results_filtered <- go_results[go_results$qvalue < 0.05, ]
```

Vamos a visualizar los términos obtenidos en un dotplot que nos permita ver la significancia de los términos y el número de genes asociados a cada uno de ellos.

```r
library(enrichplot)

## Warning: package 'enrichplot' was built under R version 4.4.2

## enrichplot v1.26.6 Learn more at https://yulab-smu.top/contribution-
knowledge-mining/
##
## Please cite:
##
## Guangchuang Yu, Li-Gen Wang, Guang-Rong Yan, Qing-Yu He. DOSE: an
## R/Bioconductor package for Disease Ontology Semantic and Enrichment
## analysis. Bioinformatics. 2015, 31(4):608-609

# Dotplot
dotplot(go_results, showCategory = 20) +
  ggtitle("Top 20 Términos de GO (Biological Process) - COVID-19 vs
Healthy")
```



También podemos visualizar los resultados en un heatmap para ver la relación entre los términos y los genes asociados a cada uno de ellos.

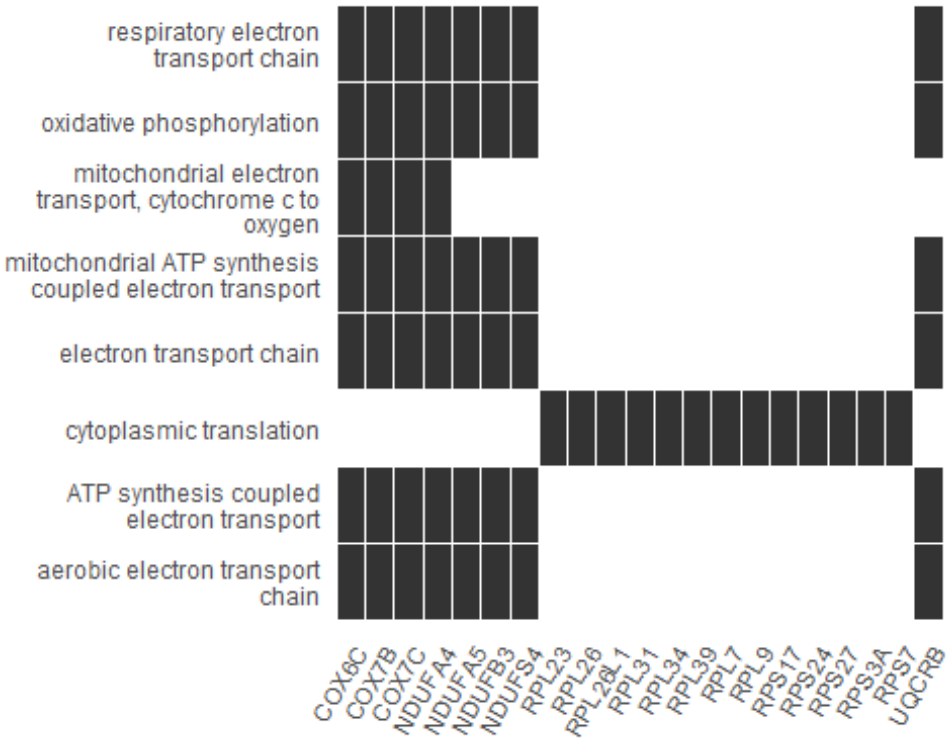
```
go_results_filtered
```

```
##                    ID
Description
## GO:0002181 GO:0002181                                    cytoplasmic
translation
## GO:0019646 GO:0019646                          aerobic electron
transport chain
## GO:0042773 GO:0042773                 ATP synthesis coupled electron
transport
## GO:0042775 GO:0042775   mitochondrial ATP synthesis coupled electron
transport
## GO:0022904 GO:0022904                      respiratory electron
transport chain
## GO:0006123 GO:0006123 mitochondrial electron transport, cytochrome c
to oxygen
## GO:0022900 GO:0022900                                   electron
transport chain
## GO:0006119 GO:0006119                                  oxidative
phosphorylation
##            GeneRatio   BgRatio RichFactor FoldEnrichment   zScore
pvalue
## GO:0002181    13/127 158/12025 0.08227848       7.790541 8.876706
1.108157e-08
```

```
## GO:0019646     8/127  84/12025 0.09523810       9.017623 7.618233
2.761456e-06
## GO:0042773     8/127  92/12025 0.08695652       8.233482 7.195415
5.494786e-06
## GO:0042775     8/127  92/12025 0.08695652       8.233482 7.195415
5.494786e-06
## GO:0022904     8/127 112/12025 0.07142857       6.763217 6.330715
2.353786e-05
## GO:0006123     4/127  18/12025 0.22222222      21.041120 8.790839
3.237573e-05
## GO:0022900     8/127 119/12025 0.06722689       6.365381 6.076877
3.650344e-05
## GO:0006119     8/127 134/12025 0.05970149       5.652838 5.595650
8.500487e-05
##               p.adjust       qvalue
## GO:0002181 0.000017254 1.718226e-05
## GO:0019646 0.002138845 2.129953e-03
## GO:0042773 0.002138845 2.129953e-03
## GO:0042775 0.002138845 2.129953e-03
## GO:0022904 0.007329689 7.299214e-03
## GO:0006123 0.008119408 8.085650e-03
## GO:0022900 0.008119408 8.085650e-03
## GO:0006119 0.016544074 1.647529e-02
##
geneID
## GO:0002181
RPS24/RPL31/RPL26/RPS7/RPL23/RPL34/RPS17/RPS3A/RPL7/RPS27/RPL9/RPL26L1/RP
L39
## GO:0019646
COX6C/COX7B/NDUFA5/NDUFS4/COX7C/UQCRB/NDUFA4/NDUFB3
## GO:0042773
COX6C/COX7B/NDUFA5/NDUFS4/COX7C/UQCRB/NDUFA4/NDUFB3
## GO:0042775
COX6C/COX7B/NDUFA5/NDUFS4/COX7C/UQCRB/NDUFA4/NDUFB3
## GO:0022904
COX6C/COX7B/NDUFA5/NDUFS4/COX7C/UQCRB/NDUFA4/NDUFB3
## GO:0006123
COX6C/COX7B/COX7C/NDUFA4
## GO:0022900
COX6C/COX7B/NDUFA5/NDUFS4/COX7C/UQCRB/NDUFA4/NDUFB3
## GO:0006119
COX6C/COX7B/NDUFA5/NDUFS4/COX7C/UQCRB/NDUFA4/NDUFB3
##            Count
## GO:0002181    13
## GO:0019646     8
## GO:0042773     8
## GO:0042775     8
## GO:0022904     8
## GO:0006123     4
```

```
## GO:0022900     8
## GO:0006119     8
```

```
heatplot(go_results, foldChange = NULL)
```



De esta tabla podemos obtener los términos GO y sus procesos biológicos celulares relacionados. Se revela una serie de procesos biológicos enriquecidos que ofrecen una posible explicación sobre los mecanismos afectados por la enfermedad. Dentro de los términos más destacados, encontramos funciones relacionadas con la traducción citoplasmática, el transporte de electrones, la fosforilación oxidativa y la regulación de la apoptosis. Sin embargo, es importante evaluar críticamente si estos hallazgos son consistentes con lo que se conoce sobre la fisiopatología de COVID-19 y cómo pueden estar influenciados por la naturaleza del análisis.

Uno de los principales hallazgos es la activación de la traducción citoplasmática, lo cual es coherente con la biología del virus SARS-CoV-2. Dado que los virus dependen de la maquinaria celular para replicarse, es lógico que las células infectadas aumenten la producción de proteínas. Este proceso podría estar impulsado no solo por la replicación viral, sino también por una respuesta celular a la infección, en la cual se sintetizan proteínas involucradas en inmunidad y reparación del daño (de Breyne et al., 2020). Sin embargo, estudios previos han sugerido que el virus también tiene mecanismos para secuestrar la traducción y evitar la síntesis de proteínas antivirales, por lo que sería interesante evaluar si esta activación de traducción citoplasmática se observa en células infectadas directamente o en células inmunitarias que responden al virus (Zhang et al., 2022).

Otro grupo de términos enriquecidos está relacionado con la producción de energía y el transporte de electrones mitocondrial. La síntesis de ATP acoplada al transporte de electrones y la fosforilación oxidativa son procesos esenciales para la función celular, y su alteración puede estar relacionada con el daño mitocondrial reportado en pacientes con COVID-19 severo (Molnar et al., 2024). La sobre-representación de estos términos podría sugerir que las células afectadas intentan compensar el daño mitocondrial, incrementando la actividad energética para responder al estrés celular. Sin embargo, algunas investigaciones han indicado que el virus puede inducir disfunción mitocondrial, lo que afectaría la producción de ATP en ciertas células inmunitarias (Noonong et al., 2023). Esto genera una posible contradicción: si el daño mitocondrial es severo, esperaríamos una reducción en la fosforilación oxidativa en lugar de un aumento.

Finalmente, los resultados muestran términos relacionados con la regulación de apoptosis y la señalización por p53, lo que sugiere una activación de mecanismos de muerte celular programada. La vía de p53 es una de las principales reguladoras de la apoptosis en respuesta al daño celular, y su activación podría estar asociada con el estrés oxidativo, el daño al ADN y la inflamación generada por la infección viral (Wang et al., 2023). Sin embargo, algunos estudios han sugerido que el virus inhibe la apoptosis en células infectadas para prolongar su replicación, lo que plantea la posibilidad de que este enriquecimiento se deba a células inmunitarias activadas en lugar de células directamente infectadas (El-Deiry & Zhang, 2024).

En conclusión, los resultados del análisis de sobrerepresentación reflejan varios procesos biológicos que tienen sentido en el contexto de COVID-19, pero también presentan algunas inconsistencias que deben evaluarse con más detalle. La activación de la traducción, el metabolismo energético y la apoptosis son características esperables en respuesta a la infección, pero es posible que estos efectos sean heterogéneos entre diferentes tipos celulares.

8. Conclusiones

En este trabajo hemos llevado a cabo un análisis de expresión diferencial a partir de los datos generados por McClain et al. (2021) en el estudio de la respuesta inmune al SARS-CoV-2. Hemos realizado un análisis exhaustivo siguiendo los pasos descritos en la introducción, y hemos obtenido los siguientes resultados:

- Hemos filtrado los datos de expresión y hemos seleccionado un subconjunto de 75 muestras aleatorias para el análisis.

- Hemos realizado un análisis exploratorio de los datos mediante PCA, MDS y HCA, y hemos identificado las variables confusoras que pueden influir en el análisis de expresión diferencial.

- Hemos llevado a cabo un análisis de expresión diferencial utilizando el método edgeR, y hemos obtenido un número significativo de genes diferencialmente expresados en los contrastes Bacterial vs healthy y COVID19 vs healthy.

- Hemos visualizado los resultados del análisis de expresión diferencial y el análisis de enriquecimiento de GO mediante gráficos y heatmaps.

- Hemos identificado un número significativo de genes que son diferencialmente expresados en ambos contrastes, lo que sugiere que hay una superposición en los perfiles de expresión génica entre las muestras Bacterial y COVID19.

- Hemos identificado un número significativo de términos enriquecidos en los genes diferencialmente expresados, lo que sugiere que hay procesos biológicos alterados en los pacientes con COVID19 en comparación con los controles sanos.

Los resultados de este análisis son coherentes con lo que se conoce sobre la fisiopatología de COVID-19, pero también presentan algunas inconsistencias que deben evaluarse con más detalle. En general, los resultados sugieren que hay una activación de la traducción, el metabolismo energético y la apoptosis en respuesta a la infección, pero es posible que estos efectos sean heterogéneos entre diferentes tipos celulares.

9. Referencias

- de Breyne, S., Vindry, C., Guillin, O., et al. (2020). Translational control of coronaviruses. Nucleic Acids Research, 48(22), 12502–12522. https://doi.org/10.1093/nar/gkaa601

- El-Deiry, W. S., & Zhang, S. (2024). SARS-CoV-2 spike protein disrupts p53 tumor suppressor pathway. Oncotarget, 15. https://doi.org/10.18632/oncotarget.28576

- Molnar, T., Lehoczki, A., Fekete, M., et al. (2024). Mitochondrial dysfunction in long COVID: mechanisms, consequences, and potential therapeutic approaches. GeroScience, 46, 5267–5286. https://doi.org/10.1007/s11357-024-01165-5

- Noonong, K., Chatatikun, M., Surinkaew, S., et al. (2023). Mitochondrial oxidative stress, mitochondrial ROS storms in long COVID pathogenesis. Frontiers in Immunology, 14, 1275001. https://doi.org/10.3389/fimmu.2023.1275001

- Wang, X., Liu, Y., Li, K., & Hao, Z. (2023). Roles of p53-mediated host–virus interaction in coronavirus infection. International Journal of Molecular Sciences, 24(7), 6371. https://doi.org/10.3390/ijms24076371

- Zhang, D., Zhu, L., Wang, Y., et al. (2022). Translational control of COVID-19 and its therapeutic implication. Frontiers in Immunology, 13, 857490. https://doi.org/10.3389/fimmu.2022.857490